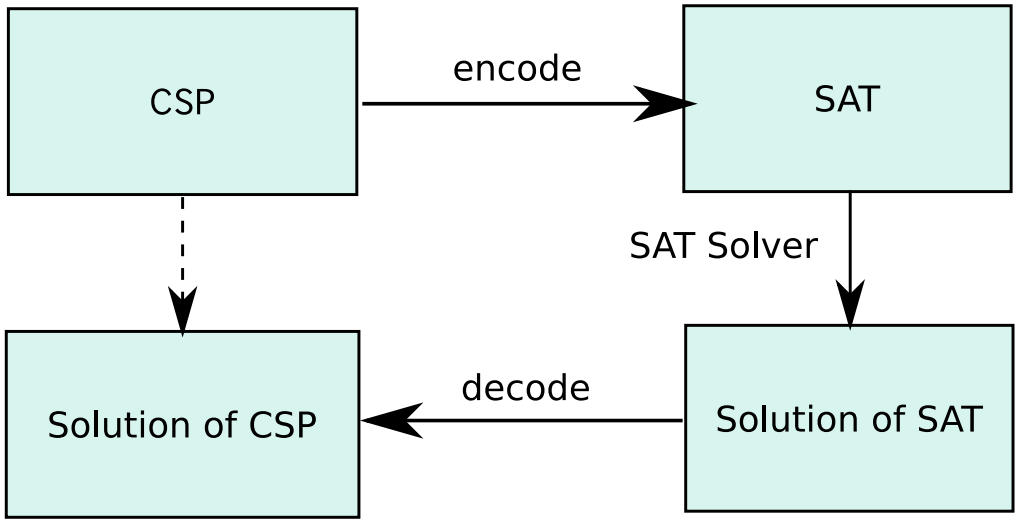
Domain-Specific Language Copris for Constraint Programming in Scala

Naoyuki Tamura (Kobe University)
Tomoya Tanjo (NII)
Mutsunori Banbara (Kobe University)

The CRIL-NII Collaborative Meeting
(November 23rd, 2012 at Université d'Artois)

Sugar: a SAT-based Constraint Solver

SAT-based Constraint Solver Sugar



- Winner of 2008 and 2009 CSP Solver Competitions in GLOBAL categories and 2008 Max-CSP Solver Competition in three categories (thanks to organizers)
- It has shown good performance in various problems (Shop Scheduling, 2D Strip Packing, Test Case Generation, Timetabling, etc.).
- Based on an Order Encoding [Tamura et al., Constraints 2009]

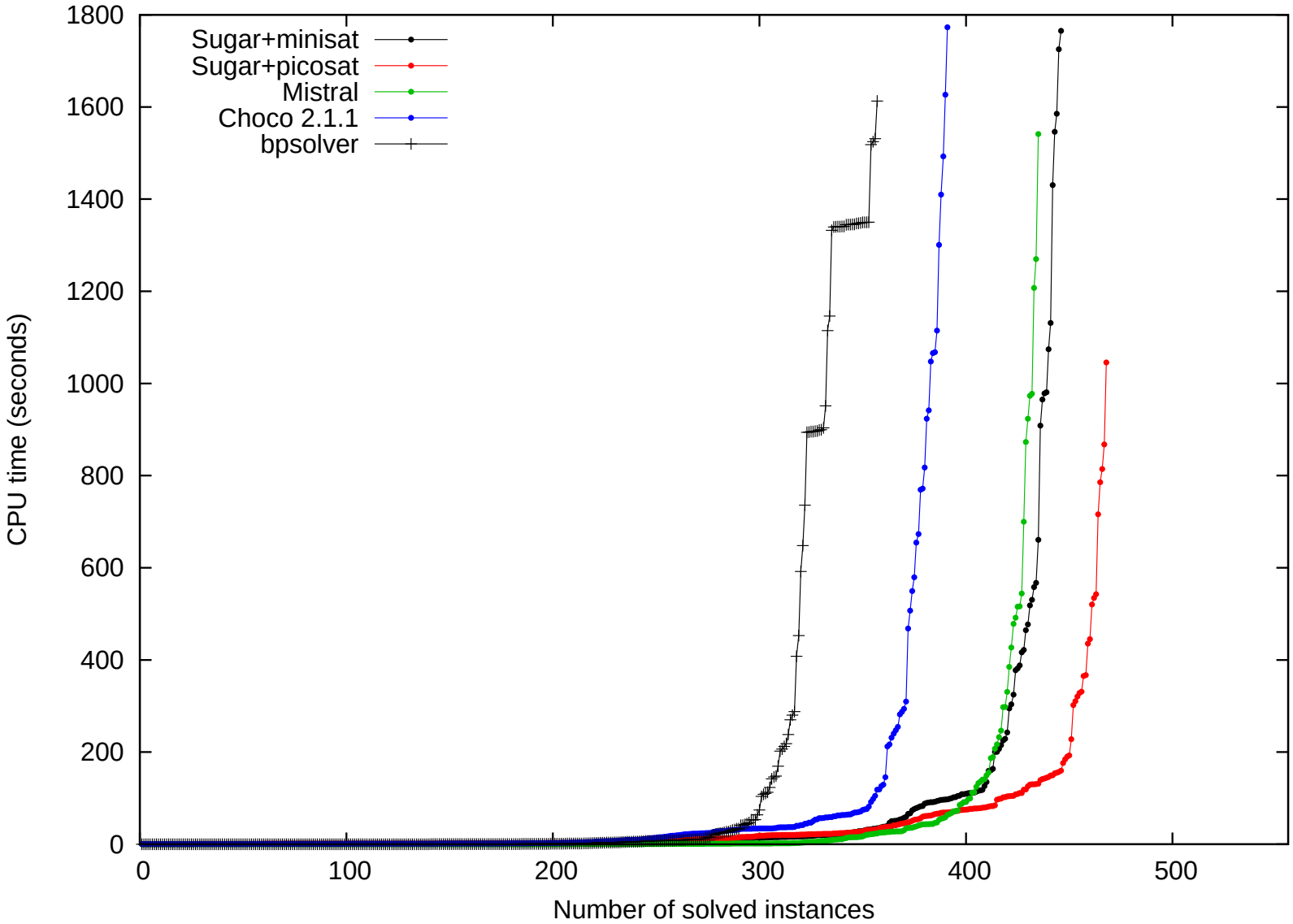
Results of GLOBAL (556 instances)

GLOBAL: Comparison of Sugar with top solvers

Series		Sugar+m	Sugar+p	Mistral	Choco	bpsolver
BIBD	(83)	76	77	76	58	35
Costas Array	(11)	8	8	9	9	9
Latin Square	(10)	10	9	5	5	5
Magic Square	(18)	8	8	13	15	11
NengFa	(3)	3	3	3	3	3
Orthogonal Latin Square	(9)	3	3	3	2	3
Perfect Square Packing	(74)	54	53	40	47	36
Pigeons	(19)	19	19	19	19	19
Quasigroup Existence	(35)	30	29	29	28	30
Pseudo-Boolean	(100)	68	75	59	53	70
BQWH	(20)	20	20	20	20	20
Cumulative Job-Shop	(10)	4	4	2	1	0
RCPSP	(78)	78	78	78	77	75
Cabinet	(40)	40	40	40	40	40
Timetabling	(46)	25	42	39	14	1
Total	(556)	446	468	435	391	357

- Combined results of three GLOBAL categories
- Sugar+m : Sugar with MiniSat 2.0 backend
- Sugar+p : Sugar with PicoSAT 535 backend

GLOBAL: Time vs Number of solved instances



Results of 2-ARY-INT (686 instances)

2-ARY-INT: Comparison of Sugar with top solvers

Series		Sugar+m	Sugar+p	Abscon	Mistral	Choco
Domino	(10)	10	10	10	10	10
Knights	(10)	6	5	8	7	4
Langford	(43)	26	26	27	30	29
NengFa	(3)	3	3	3	3	3
Pigeons	(10)	4	4	10	10	10
Queen Attacking	(10)	4	4	3	4	4
Queens	(15)	10	10	11	10	12
Queens-Knights	(10)	10	10	9	8	5
Pseudo-Boolean	(1)	1	1	1	1	1
Graph Coloring	(141)	95	93	119	121	118
Haystacks	(15)	3	2	6	3	6
Job-Shop	(76)	66	66	45	49	62
Open-Shop	(75)	71	71	47	70	75
Super-solutions	(85)	60	60	40	52	49
Crossword	(3)	3	3	3	3	3
FAPP	(120)	39	52	116	73	63
RLFAP	(59)	59	59	59	57	56
Total	(686)	470	479	517	511	510

- Sugar+m : Sugar with MiniSat 2.0 backend
- Sugar+p : Sugar with PicoSAT 535 backend

Results of N-ARY-INT (709 instances)

N-ARY-INT: Comparison of Sugar with top solvers

Series		Sugar+m	Sugar+p	Mistral	Choco	pcs-r
All Interval Series	(15)	9	9	12	11	15
Chessboard Coloration	(15)	11	11	12	10	11
Golomb Ruler	(28)	19	17	21	23	23
NengFa	(4)	4	4	3	3	4
Ramsey	(16)	10	9	10	10	5
Schurr's Lemma	(10)	8	8	8	7	8
Social Golfers	(10)	6	5	6	5	5
Pseudo-Boolean	(262)	189	201	190	180	179
2D Strip Packing	(20)	7	7	5	3	3
Fischer	(25)	14	14	13	15	17
Multi Knapsack	(6)	4	4	5	6	4
Primes	(76)	43	44	70	70	57
Rader Surveillance	(65)	65	65	65	65	65
BMC	(15)	15	15	15	15	15
Crossword	(142)	77	65	137	137	135
Total	(709)	481	478	572	560	546

- Sugar+m : Sugar with MiniSat 2.0 backend
- Sugar+p : Sugar with PicoSAT 535 backend

Order Encoding

In order encoding (also known as regular encoding or ladder encoding), a Boolean variable $p(x \leq i)$ is defined as true iff the integer variable x is less than or equal to the domain value i, that is, $x \leq i$.

Example of encoding $x \in \{2, 3, 4, 5, 6\}$

$$\neg p(x \leq 2) \vee p(x \leq 3)$$
$$\neg p(x \leq 3) \vee p(x \leq 4)$$
$$\neg p(x \leq 4) \vee p(x \leq 5)$$

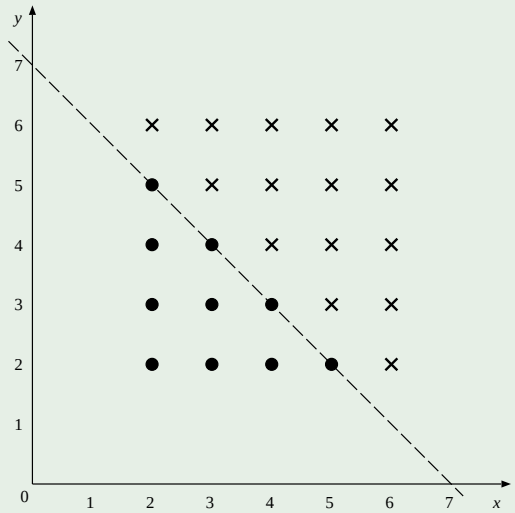
- Partial assignments on Boolean variables represent bounds of integer variables.

$p(x \leq 2)$	$p(x \leq 3)$	$p(x \leq 4)$	$p(x \leq 5)$	Intepretation
0	–	1	1	$x = 3..4$

Order Encoding

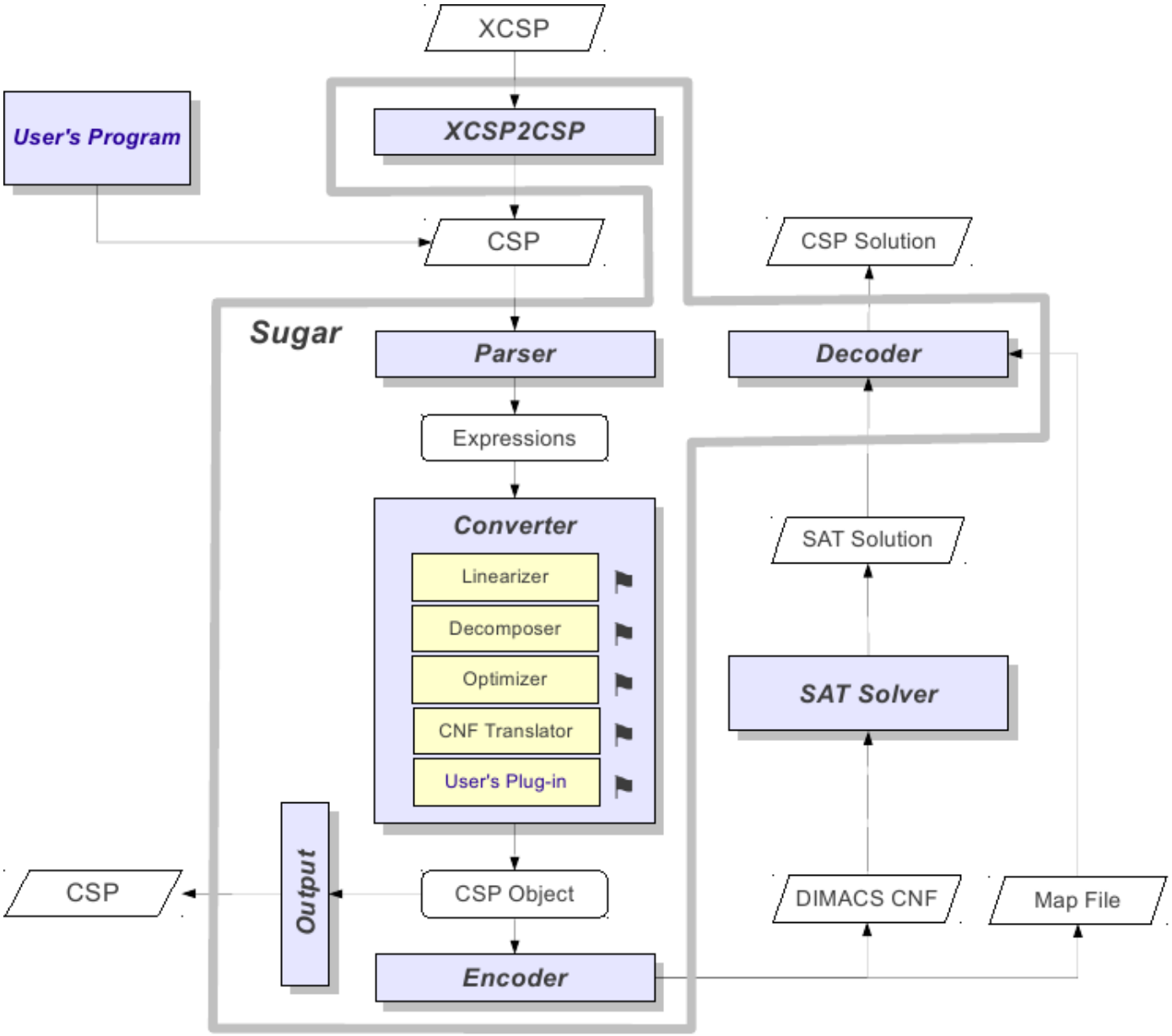
Example of encoding of $x + y \leq 7$ when $x, y \in \{2..6\}$

$$
\begin{aligned}
C_1 &: \quad p(y \leq 5) \\
C_2 &: \quad p(x \leq 2) \vee p(y \leq 4) \\
C_3 &: \quad p(x \leq 3) \vee p(y \leq 3) \\
C_4 &: \quad p(x \leq 4) \vee p(y \leq 2) \\
C_5 &: \quad p(x \leq 5)
\end{aligned}
$$



- Suppose $p(x \leq 3)$ becomes false (i.e. $x \geq 4$), then $p(y \leq 3)$ becomes true (i.e. $y \leq 3$) by unit propagation on C_3.
- This corresponds to the bound propagation in CSP solvers.
- It is shown that the order encoindg is the only SAT-encoding translating tractable CSP to tractable SAT [Petke and Jeavons 2011].

Architecture of Sugar



CSP Modeling Lanuage of Sugar

- List notations in Lisp-like syntax is used (XCSP is covered).
- No arrays, no loops. So, it is difficult to write a large problem by hand.
- Users usally need to write a program (e.g. in Perl) generating the input file to Sugar.

Example of the Sugar input file for 4-Queens

```
(int q_1 1 4)
(int q_2 1 4)
(int q_3 1 4)
(int q_4 1 4)
(alldifferent q_1 q_2 q_3 q_4)
(alldifferent (+ q_1 1) (+ q_2 2) (+ q_3 3) (+ q_4 4))
(alldifferent (- q_1 1) (- q_2 2) (- q_3 3) (- q_4 4))
```

Copris: DSL for Constraint Programming in Scala [1]

[1] DSL: Domain-Specific Language

Outline of Scala

- New programming language designed by Martin Odersky
- Features of Scala language
  - Integration of functional and object-oriented programming paradigms
  - Type inferences, higher order functions
  - Immutable Collections
  - Concurrent computation (Actor model)
  - Suitable for implementing DSL embedded in Scala
- Features of Scala system
  - Compiler system generating JVM [2] code
    - Java class libraries can be used
  - Interactive environment (REPL [3])

[2] JVM: Java Virtual Machine
[3] REPL: Read Eval Print Loop

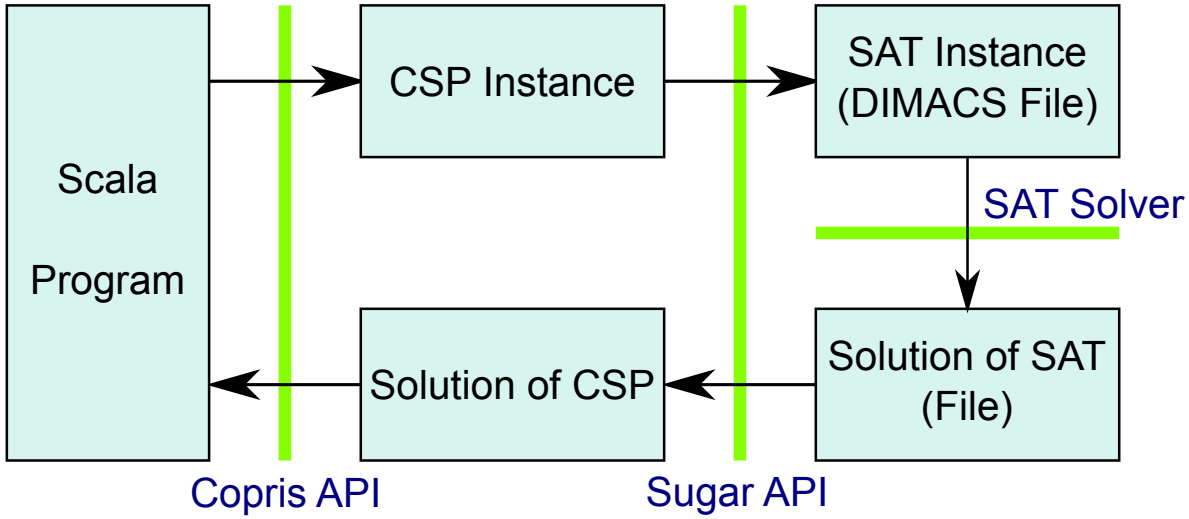
Copris (Constraint Programming in Scala)

Copris is a Domain-Specific Langauge for Constraint Programming embedded in Scala

Features

- It provides easy-to-use CSP modeling functions.
- It realizes high-performance constraint solving by using Sugar.
- When using Sat4j as its backend SAT solver, everything runs on JVM.
- It can be used as a generator of SAT instances.

Architecture of Copris



1. CSP instance is defined thru Copris API from user's Scala program.
2. Copris solver is called from user's Scala program.
   - CSP instance is encoded by Sugar to a SAT instance file.
   - SAT solver (Sat4j is the default) is called from Copris solver. User can specifiy other SAT solvers such as Glucose, GlueMiniSat, MiniSat, etc.
   - SAT solver's solution is decoded to a CSP solution by Sugar.
3. The CSP solution is returned back to user's program.

Example of Copris: FirstStep.scala

There are some cranes and tortoises. The total number is 7, and the total number of legs is 20. How many cranes and tortoises?

Example of Copris: FirstStep.scala

There are some cranes and tortoises. The total number is 7, and the total number of legs is 20. How many cranes and tortoises?

```
 1:  import jp.kobe_u.copris._
 2:  import jp.kobe_u.copris.dsl._
 3:
 4:  object FirstStep {
 5:    def main(args: Array[String]) = {
 6:      int('x, 0, 7)
 7:      int('y, 0, 7)
 8:      add('x + 'y === 7)
 9:      add('x * 2 + 'y * 4 === 20)
10:      if (find) {
11:        println(solution)
12:      }
13:    }
14:  }
```

Copris DSL: Imports

```
import jp.kobe_u.copris._
import jp.kobe_u.copris.dsl._
```

- First line imports classes provided by Copris.
- Second line imports DSL methods provided by Copris.
  - `int(x, lb, ub)` method defines an integer variable.
  - `add(c)` method defines a constraint.
  - `find` method searches a solution.
  - `solution` method returns the solution.

Copris DSL: Integer variables

```
int('x, 0, 7)      // Declare x ∈ {0..7}
int('y, Set(0,2)) // Declare y ∈ {0, 2}
```

- `'x` is a notation of symbols in Scala, and they are automatically converted integer variable (`Var`) objects objeof Copris by an implicit conversion defiend in Copris.
- Integer variables declared by `int` methods are added to the default CSP predefined in Copris.

```
int('a(1), 0, 7)     // Declare a(1) ∈ {0..7}
int('b(1,2), 0, 7)  // Declare b(1,2) ∈ {0..7}
```

- When some indices are given to an integer variable object, new integer variable object is created.

Copris DSL: Constraints

```
add('x + 'y === 7)
add('x < 'y || 'y >= 'z)
add(Alldifferent('x, 'y, 'z))
```

- Arithmetic and logical operators can be used to represent constraints.
  - Equality ===, non-equality !==, implication ==>
- Global constraints can be represented by using corresponding class names.
- Constraints declared by add methods are added to the default CSP predefined in Copris.

Copris DSL: Solver

```
find      // finds the first solution
findNext  // finds the next solution
findOpt   // finds the optimum solution
```

- The `find` method encodes the defined CSP to SAT, and invokes SAT solver to find a solution.
- The `findNext` method adds the negation of the current solution to the CSP (also adds to the genrated SAT instacne), and invokes SAT solver again to find another solution.
- The `findOpt` method finds the optimum solution by dichotomic search.

Example of Copris: N-Queens

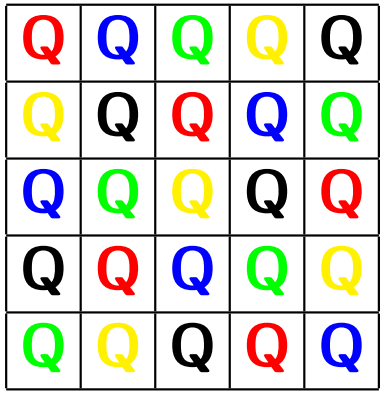
Place n queens on 8×8 chess board such that any queen is not captured by others.

```
 1:  val n = 8
 2:  for (i <- 1 to n) int('q(i), 1, n)
 3:  add(Alldifferent((1 to n).map(i => 'q(i))))
 4:  add(Alldifferent((1 to n).map(i => 'q(i)+i)))
 5:  add(Alldifferent((1 to n).map(i => 'q(i)-i)))
 6:  if (find) {
 7:    do {
 8:      println(solution)
 9:    } while (findNext)
10:  }
```

- `'q(i)` is an integer variable representing the column number of the queen at the i-th row.

Example of Copris: Colored Queens

Place n differently colored sets of n queens on a $n \times n$ board so that no two queens of the same color attack each other.

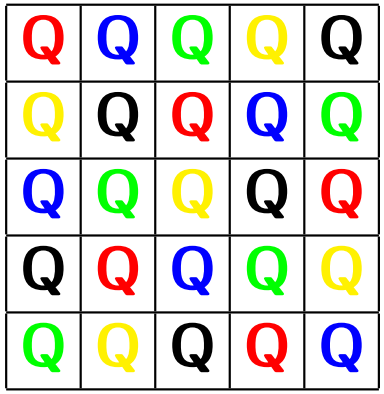


Pólya described the problem has a solution iff $n \equiv \pm 1 \pmod 6$.

n	5	6	7	8	9	10	11	12	13
SAT/UNSAT	S	U	S	U	U	U	S	U	S

Example of Copris: Colored Queens

Place n differently colored sets of n queens on a $n \times n$ board so that no two queens of the same color attack each other.



Pólya described the problem has a solution iff $n \equiv \pm 1 \pmod 6$.

n	5	6	7	8	9	10	11	12	13
SAT/UNSAT	S	U	S	U	U	U	S	S	S

However, Patrick Hamlyn and Guenter Stertenbrink found a solution when $n = 12$.

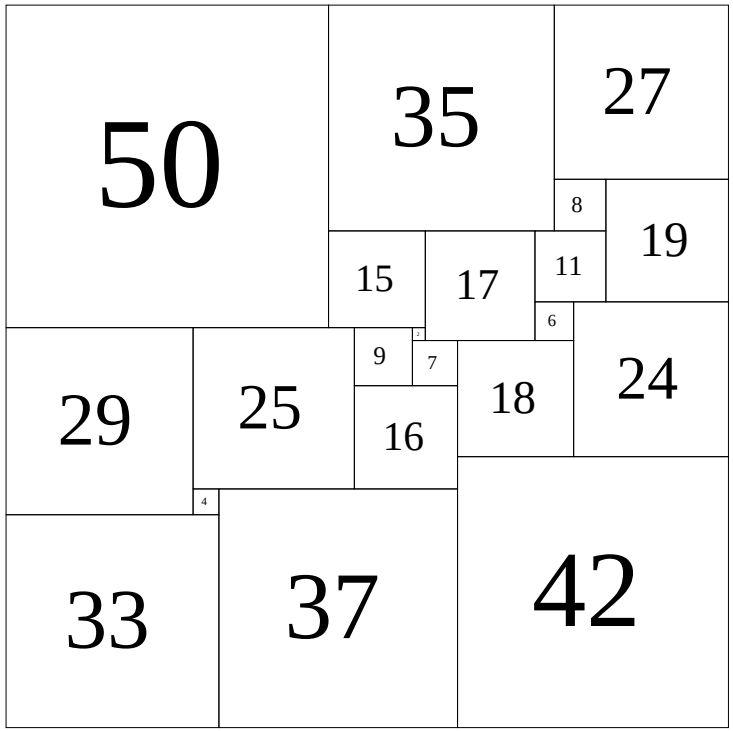
Example of Copris: Colored Queens

```
1:  for (c <- 1 to n; i <- 1 to n)
2:    int('q(c,i), 1, n)
3:  for (i <- 1 to n)
4:    add(Alldifferent((1 to n).map(c => 'q(c,i))))
5:  for (c <- 1 to n) {
6:    add(Alldifferent((1 to n).map(i => 'q(c,i))))
7:    add(Alldifferent((1 to n).map(i => 'q(c,i) + i)))
8:    add(Alldifferent((1 to n).map(i => 'q(c,i) - i)))
9:  }
```

- Copris can find solutions for all $n \leq 13$ except $n = 12$.
  Any helps are welcome!
- It will be interesting to find another exception of $n > 13$.

Example of Copris: Perfect Square Packing

Packing 21 squares of the size 2, 4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 24, 25, 27, 29, 33, 35, 37, 42, 50 to 112×112 square.



21 : 112A AJD 1978

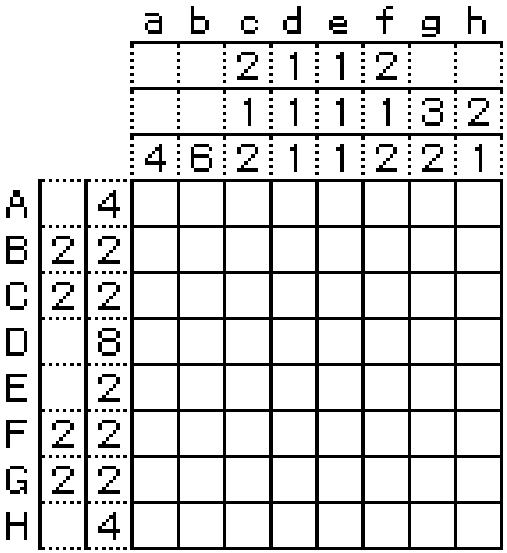
Example of Copris: Perfect Square Packing

```
 1:  val size = 112
 2:  val s = List(2,4,6,7,8,9,11,15,16,17,
 3:    18,19,24,25,27,29,33,35,37,42,50)
 4:  val n = s.size
 5:  for (i <- 0 to n-1) {
 6:    int('x(i), 0, size - s(i))
 7:    int('y(i), 0, size - s(i))
 8:  }
 9:  for (i <- 0 to n-1; j <- i+1 to n-1)
10:    add('x(i) + s(i) <= 'x(j) ||
11:        'x(j) + s(j) <= 'x(i) ||
12:        'y(i) + s(i) <= 'y(j) ||
13:        'y(j) + s(j) <= 'y(i))
```

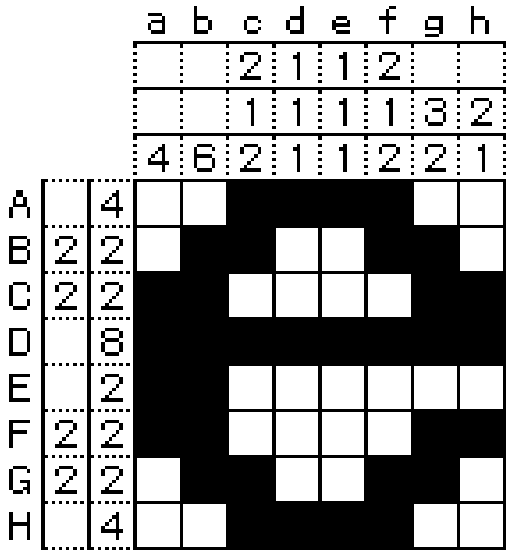
- `'x(i)` and `'y(i)` represent the bottom left position of the i-th square.
- Lines 10-13 represent the condition such that the i-th and j-th squares can not be overlapped.
- Copris+MiniSat can find a solution in 60 seconds.

Example of Copris: Nonogram

問題



解答



- Copris can solve all problems in Survey of Paint-by-Number Puzzle Solvers (http://webpbn.com/survey/) while other solvers listed do not.
- It can solve much larger problem, e.g. 100×100 sized problem in http://www.comp.lancs.ac.uk/~ss/nonogram/puzzles.

Example of Copris: Nonogram

puzzle	size	Copris Sat4j	Wolter	Olsak	Simpson	BGU	Lagerkvist Gecode	Kjellerstrand Gecode	Kjellerstrand Lazyfd
1: Dancer	5 x 10	0.60	0.00	0.00	0.00	0.06	0.01	0.01	0.04
6: Cat	20 x 20	2.73	0.00	0.00	0.00	0.08	0.01	0.02	0.44
21: kid	14 x 25	2.91	0.00	0.00	0.00	0.08	0.01	0.02	0.64
27: Buck	27 x 23	4.69	0.00	0.00	0.00	0.16	0.01	0.02	1.10
23: Edge	10 x 11	0.86	0.00	0.00	0.00	0.09	0.01	0.01	0.08
2413: Smoke	20 x 20	5.54	0.00	0.00	0.00	0.19	0.01	0.02	0.60
16: Knot	34 x 34	6.36	0.00	0.00	0.00	0.20	0.01	0.02	5.50
529: Swing	45 x 45	8.99	0.00	0.00	0.00	0.25	0.02	0.04	13.20
65: Mum	34 x 40	8.88	0.00	0.00	0.01	0.26	0.02	0.04	11.00
7604: DiCap	52 x 63	17.88	0.00	0.00	0.00	2.10	0.02	0.06	+
1694: Tragic	45 x 50	9.64	0.00	0.03	0.54	0.62	0.14	216.00	32.10
1611: Merka	55 x 60	11.35	0.01	0.01	0.00	0.35	0.03	+	66.00
436: Petro	40 x 35	8.06	0.06	15.20	0.10	0.58	84.00	1.60	6.20
4645: M&M	50 x 70	10.40	0.07	0.10	0.02	0.84	0.93	0.28	48.10
3541: Signed	60 x 50	10.60	0.04	1.10	324.00	0.68	0.57	0.56	60.00
803: Light	50 x 45	8.72	0.38	+	0.02	1.10	+	+	4.70
6574: Forever	25 x 25	7.34	3.70	2.00	18.90	44.30	4.70	2.30	1.70
2040: Hot	55 x 60	12.70	0.83	+	72.00	0.93	+	+	156.00
6739: Karate	40 x 40	12.48	0.80	17.30	1098.00	0.61	56.0	57.8	13.60
8098: 9-Dom	19 x 19	4.59	11.00	240.00	+	168.00	756.00	3.80	1.80
2556: Flag	65 x 45	8.79	0.55	1.50	0.00	24.20	3.40	+	4.20
2712: Lion	47 x 47	11.65	6.30	+	+	7.80	+	+	+
10088: Marley	52 x 63	25.95	+	+	+	22.00	+	+	174.00
9892: Nature	50 x 40	16.02	+	1116.00	+	+	+	+	144.00

- Copris+Sat4j: Intel Core i5 vPro 2.6GHz
- Other solvers: AMD Phenom II X4 810 2.6GHz
- "+" means the timeout (1800 seconds)

Example of Copris: Nonogram

```
 1:  for (i <- 0 until m; j <- 0 until n)
 2:    int('x(i,j), 0, 1)
 3:  for (i <- 0 until m; k <- 0 until rows(i).size)
 4:    int('r(i,k), 0, n-rows(i)(k))
 5:  for (i <- 0 until m; k <- 0 until rows(i).size-1)
 6:    add('r(i,k) + rows(i)(k) < 'r(i,k+1))
 7:  for (j <- 0 until n; k <- 0 until cols(j).size)
 8:    int('c(j,k), 0, m-cols(j)(k))
 9:  for (j <- 0 until n; k <- 0 until cols(j).size-1)
10:    add('c(j,k) + cols(j)(k) < 'c(j,k+1))
11:  for (i <- 0 until m; j <- 0 until n) {
12:    val rs = for (k <- 0 until rows(i).size)
13:             yield 'r(i,k) <= j && 'r(i,k) + rows(i)(k) > j
14:    add('x(i,j) > 0 <==> Or(rs))
15:    val cs = for (k <- 0 until cols(j).size)
16:             yield 'c(j,k) <= i && 'c(j,k) + cols(j)(k) > i
17:    add('x(i,j) > 0 <==> Or(cs))
18:  }
```

- `'x(i,j)` indicates the color of the cell at (i, j) position (1 means black).
- `'r(i,k)` indicates the left most position of the k-th block at i-th line.
- `'c(j,k)` indicates the upper most position of the j-th block at j-th row.

Summray

- We introduced
  - Sugar (a SAT-based Constraint Solver) and
  - Copris (a DSL for Constraint Programming in Scala).

Future works:

- Better integration with Sat4j
  - Incremental SAT solving
  - Enumerating solutions
- Enhancement of Copris for soft constraints (Max-CSP, Weighted CSP, etc.)
- Using Azucar (new version of Sugar) as backend solver